

Programación Concurrente

Teoría 8



Facultad de Informática
UNLP

Links al archivo con audio

La teoría con los audios está en formato MP4. Debe descargar los archivos comprimidos de los siguientes links:

◆ Paradigmas de Interacción entre Procesos:

<https://drive.google.com/uc?id=1a0QUfXjpdM26pTIGzSEjm7zGRtCxAs9d&export=download>



Paradigmas de Interacción entre Procesos

Paradigmas para la interacción entre procesos

- 3 esquemas básicos de interacción entre procesos: *productor/consumidor*, *cliente/servidor* e *interacción entre pares*.
- Estos esquemas básicos se pueden combinar de muchas maneras, dando lugar a otros **paradigmas** o modelos de interacción entre procesos.

Paradigma 1: *master / worker*

Implementación distribuida del modelo *Bag of Task*.

Paradigma 2: *algoritmos heartbeat*

Los procesos periódicamente deben intercambiar información con mecanismos tipo send/receive.

Paradigma 3: *algoritmos pipeline*

La información recorre una serie de procesos utilizando alguna forma de receive/send.

Paradigmas para la interacción entre procesos

Paradigma 4: *probes (send) y echoes(receive)*

La interacción entre los procesos permite recorrer grafos o árboles (o estructuras dinámicas) disseminando y juntando información.

Paradigma 5: *algoritmos broadcast*

Permiten alcanzar una información global en una arquitectura distribuida. Sirven para toma de decisiones descentralizadas.

Paradigma 6: *token passing*

En muchos casos la arquitectura distribuida recibe una información global a través del viaje de tokens de control o datos. También permite la toma de decisiones distribuidas.

Paradigma 7: *servidores replicados*

Los servidores manejan (mediante múltiples instancias) recursos compartidos tales como dispositivos o archivos.

Paradigmas para la interacción entre procesos

Manager/Worker

- El concepto de *bag of tasks* usando variables compartidas supone que un conjunto de workers comparten una “bolsa” con tareas independientes. Los workers sacan una tarea de la bolsa, la ejecutan, y posiblemente crean nuevas tareas que ponen en la bolsa (ejemplo en LINDA manejando un espacio compartido de tuplas).
- La mayor virtud de este enfoque es la escalabilidad y la facilidad para equilibrar la carga de trabajo de los workers.
- Analizaremos la implementación de este paradigma con mensajes en lugar de MC. Para esto un proceso *manager* implementará la “bolsa” manejando las tasks, comunicándose con los workers y detectando fin de tareas. **Se trata de un esquema C/S.**
- Ejemplo: multiplicación de matrices ralas.

Paradigmas para la interacción entre procesos

Heartbeat

- Paradigma *heartbeat* \Rightarrow útil para soluciones iterativas que se quieren paralelizar.
- Usando un esquema “*divide & conquer*” se distribuye la carga (datos) entre los workers; cada uno es responsable de actualizar una parte.
- Los nuevos valores dependen de los mantenidos por los workers o sus vecinos inmediatos.
- Cada “paso” debiera significar un progreso hacia la solución.
- Formato general de los worker:

```
process worker [i=1 to numWorkers]
{  declaraciones e inicializaciones locales;
  while (no terminado)
  {  send valores a los workers vecinos;
    receive valores de los workers vecinos;
    Actualizar valores locales;
  }
}
```

- Ejemplo: grid computations (imágenes), autómatas celulares (simulación de fenómenos como incendios o crecimiento biológico).

Paradigmas para la interacción entre procesos

Heartbeat - Topología de una red

Los procesadores están conectados por canales bidireccionales. Cada uno se comunica sólo con sus vecinos y conoce esos links.

¿Cómo puede cada procesador determinar la topología completa de la red?

➤ Modelización:

- Procesador \Rightarrow proceso
- Links de comunicación \Rightarrow canales compartidos.

➤ Soluciones: los vecinos interactúan para intercambiar información local.

Algoritmo Heartbeat: se expande enviando información; luego se contrae incorporando nueva información.

➤ Procesos *Nodo*[$p:1..n$].

➤ Vecinos de p : *vecinos*[$1:n$] \rightarrow *vecinos*[q] es true si q es vecino de p .

➤ **Problema:** computar *top* (matriz de adyacencia), donde *top*[p,q] es true si p y q son vecinos.

Paradigmas para la interacción entre procesos

Heartbeat - Topología de una red

Cada nodo debe ejecutar un n° de rondas para conocer la topología completa. Si el diámetro D de la red es conocido se resuelve con el siguiente algoritmo.

```
chan topologia[1:n] ([1:n,1:n] bool)

Process Nodo[p:1..n]
{ bool vecinos[1:n], bool nuevatop[1:n,1:n], top[1:n,1:n] = ([n*n] false);
  top[p,1..n] = vecinos;

  for (r = 0 ; r < D; r++)
    { for [q = 1 to n st vecinos[q] ] send topologia[q](top);
      for [q = 1 to n st vecinos[q] ]
        { receive topologia[p](nuevatop);
          top = top or nuevatop;
        }
    }
}
```

Paradigmas para la interacción entre procesos

Heartbeat - Topología de una red

- Rara vez se conoce el valor de D .
- Excesivo intercambio de mensajes \Rightarrow los procesos cercanos al “centro” conocen la topología más pronto y no aprenden nada nuevo en los intercambios.
- El tema de la terminación \Rightarrow ¿local o distribuida?
- *¿Cómo se pueden solucionar estos problemas?*
 - Después de r rondas, p conoce la topología a distancia r de él. Para cada nodo q dentro de la distancia r de p , los vecinos de q estarán almacenados en la fila q de $top \Rightarrow p$ ejecutó las rondas suficientes tan pronto como cada fila de top tiene algún valor *true*.
 - Luego necesita ejecutar una última ronda para intercambiar la topología con sus vecinos.
- No siempre la terminación se puede determinar localmente.

Paradigmas para la interacción entre procesos

Heartbeat - Topología de una red

```
chan topologia[1:n](emisor : int; listo : bool; top : [1:n,1:n] bool)

Process Nodo[p:1..n]
{ bool vecinos[1:n], activo[1:n] = vecinos, top[1:n,1:n] = ([n*n]false), nuevatop[1:n,1:n];
  bool qlisto, listo = false;
  int emisor;
  top[p,1..n] = vecinos;
  while (not listo)
  {   for [q = 1 to n st activo[q] ] send topologia[q](p,false,top);
      for [q = 1 to n st activo[q] ]
      {   receive topologia[p](emisor,qlisto,nuevatop);
          top = top or nuevatop;
          if (qlisto) activo[emisor] = false;
      }
      if (todas las filas de top tiene 1 entry true) listo=true;
  }
  for [q = 1 to n st activo[q] ] send topologia[q](p,listo,top);
  for [q=1 to n st activo[q]] receive topologia[p](emisor,d,nuevatop);
}
```

Paradigmas para la interacción entre procesos

Pipeline

- Un pipeline es un arreglo lineal de procesos “filtro” que reciben datos de un puerto (canal) de entrada y entregan resultados por un canal de salida.
- Estos procesos (“workers”) pueden estar en procesadores que operan en paralelo, en un primer esquema *a lazo abierto* (W_1 en el INPUT, W_n en el OUTPUT).
- Un segundo esquema es el pipeline *circular*, donde W_n se conecta con W_1 . Estos esquemas sirven en procesos iterativos o bien donde la aplicación no se resuelve en una pasada por el pipe.
- En un tercer esquema posible (*cerrado*), existe un proceso coordinador que maneja la “realimentación” entre W_n y W_1 .
- Ejemplo: multiplicación de matrices en bloques.

Paradigmas para la interacción entre procesos

Probe-Echo

- Árboles y grafos son utilizados en muchas aplicaciones distribuidas como búsquedas en la WEB, BD, sistemas expertos y juegos.
- Las arquitecturas distribuidas se pueden asimilar a los nodos de grafos y árboles, con canales de comunicación que los vinculan.
- DFS es uno de los paradigmas secuenciales clásicos para visitar todos los nodos en un árbol o grafo. Este paradigma es el análogo concurrente de DFS.
- ***Prueba-eco*** se basa en el envío de un mensajes (“probe”) de un nodo al sucesor, y la espera posterior del mensaje de respuesta (“echo”).
- Los **probes** se envían en paralelo a todos los sucesores.
- Los algoritmos de prueba-eco son particularmente interesantes cuando se trata de recorrer redes donde no hay (o no se conoce) un número fijo de nodos activos (ejemplo: redes móviles).

Paradigmas para la interacción entre procesos

Broadcast

- En la mayoría de las LAN cada procesador se conecta directamente con los otros. Estas redes normalmente soportan la primitiva ***broadcast***:

broadcast ch(m);

- Los mensajes broadcast de un proceso se encolan en los canales en el orden de envío, pero broadcast no es atómico y los mensajes enviados por procesos A y B podrían ser recibidos por otros en distinto orden.
- Se puede usar broadcast para diseminar información o para resolver problemas de sincronización distribuida. Ejemplo: semáforos distribuidos, la base es un ***ordenamiento total de eventos de comunicación*** mediante el uso de ***relojes lógicos***.

Paradigmas para la interacción entre procesos

Token Passing

- Un paradigma de interacción muy usado se basa en un tipo especial de mensaje (“token”) que puede usarse para otorgar un permiso (control) o recoger información global de la arquitectura distribuida. Un ejemplo del primer tipo de algoritmos es el caso de tener que controlar ***exclusión mutua distribuida***.
- Ejemplos de recolección de información de estado son los algoritmos de detección de terminación en computación distribuida.
- Aunque el problema de la SC se da principalmente en programas de MC, puede encontrarse en programas distribuidos cuando hay algún recurso compartido que puede usar un único proceso a la vez. Generalmente es una componente de un problema más grande, tal como asegurar consistencia en un sistema de BD.
- Soluciones posibles: Monitor activo que da permiso de acceso (ej: locks en archivos), semáforos distribuidos (usando broadcast, con gran intercambio de mensajes), o *token ring* (descentralizado y fair).

Paradigmas para la interacción entre procesos

Servidores Replicados

- Un server puede ser replicado cuando hay múltiples instancias de un recurso: cada server maneja una instancia.
- La replicación también puede usarse para darle a los clientes la sensación de un único recurso cuando en realidad hay varios.
- Ejemplo: problema de los filósofos
 - Modelo **centralizado**: los Filósofo se comunican con **UN** proceso Mozo que decide el acceso o no a los recursos.
 - Modelo **distribuido**: supone **5 procesos Mozo**, cada uno manejando un tenedor. Un Filósofo puede comunicarse con **2** Mozos (izquierdo y derecho), solicitando y devolviendo el recurso. Los Mozos **NO se comunican entre ellos**.
 - Modelo **descentralizada**: cada Filósofo ve **un único** Mozo. Los Mozos se comunican entre ellos (cada uno con sus **2** vecinos) para decidir el manejo del recurso asociado a “su” Filósofo.